An Adaptable Framework for the Teaching and Assessment of Software Development across Year Levels

Richard N Thomas
Faculty of Science and Technology
Queensland University of Technology
Brisbane, Australia
+61 7 3138 2736
r.thomas@qut.edu.au

Moira Cordiner
Centre for the Advancement of Teaching and Learning
University of Tasmania
Hobart, Australia
+61 3 6223 6794
moira.cordiner@utas.edu.au

Diane Corney
Faculty of Science and Technology
Queensland University of Technology
Brisbane, Australia
+61 7 3138 9335
d.corney@qut.edu.au

Abstract

Few frameworks exist for the teaching and assessment of programming subjects that are coherent and logical. Nor are they sufficiently generic and adaptable to be used outside the particular tertiary institutions in which they were developed. This paper presents the Teaching and Assessment of Software Development (TASD) framework. We describe its development and implementation at an Australian university and demonstrate, with examples, how it has been used, with supporting data. Extracts of criteria sheets (grading rubrics) for a variety of assessment tasks are included. The numerous advantages of this new framework are discussed with comparisons made to those reported in the published literature.

Keywords: curriculum, criterion-referenced assessment.

1 Introduction

In 2006, two of the authors (Thomas and Corney) started a research project to develop a coherent and logical approach for implementing criterion-referenced assessment (CRA) in software development subjects. The goal was to improve assessment and course design in response to continual student criticisms voiced in course evaluations. Many of these criticisms were about a lack of transparency about expectations. Students wanted expectations to be *explicit*. Before we embarked on this project, we, like other academics at QUT usually gave some explicit assessment criteria to students, but the standards we used to grade the quality of their work were implicit. The criteria were often unrelated to those used in software development subjects offered by other academics, leading to a lack of cohesion across degrees, and subsequent student dissatisfaction. CRA requires that students are provided with explicit criteria and standards for assessment and that these are aligned with teaching and learning across year levels.

The intended outcome of the research was to develop resources such as examples of assessment tasks and criteria sheets that illustrated, not only how to implement the principles of CRA, but also how to do it across different year levels. Ideally this should be in a way that was transparent, manageable, practical, and genuinely reflected the nature of software development. The project, therefore, became a search for a 'levelling' framework that would allow us to make explicit to stakeholders the level of expectations of all aspects of a degree (course) across each of the years required to complete it.

Implementing CRA has been avoided for several reasons in the Faculty. The primary reason was a strongly-held belief that it was not possible to write five standards (from highest to lowest) for students' answers to problems, whether there was only one solution or multiple solutions. The other reasons were limited time to implement and the non-availability of relevant examples. With support from our Faculty and an academic staff developer in assessment (Cordiner), we developed and implemented a framework with associated resources across different year levels.

The purposes of this paper are to:

- present the Teaching and Assessment of Software Development (TASD) framework and describe how we implemented it
- illustrate how the framework is adaptable for use with all software development subjects and provide supporting data
- discuss the advantages of our framework compared to others and reflect on how it has affected our assessment practices.

2 Literature review

An Australian national survey in 2001 (Watson, Morgan, McKenzie, Roberts, & Cochrane, 2002) revealed that universities have no policies stating explicitly what year levels imply in undergraduate degrees. Expectations for students at each level (levelling) are mostly established by content-sequencing and are not informed by a teaching and assessment framework.

In our literature review, we found a variety of frameworks used in software engineering education. In the ones that focus only on assessment, the criteria for tasks are inconsistently defined and do not reflect the processes of software development. Teachers allocate marks based on their own 'internal' (implicit) standards.

Various difficulties have been found with these types of frameworks. For example, Daniels, Berglund, Pears, and Fincher, in describing the Runestone project, noted

that 'the process of agreeing on assessment criteria and the actual meaning of them was far more difficult than expected' (2004, p58). Michael (2000) observed that, when students peer graded each other's presentations using a checklist of questions linked to criteria, there was no comparability between their judgments.

Various authors (Burgess, 2005; Box, 2004; Lister and Leaney, 2003; Buck and Stucki, 2000) have used taxonomies of generic objectives/learning outcomes as teaching *and* assessment frameworks. These objectives become increasingly complex according to the application of a given hierarchy. For example, the taxonomy developed by Bloom and Krathwohl (1956) is set out under knowledge, comprehension, application, analysis, synthesis and evaluation and implies a linear view of how students learn.

One advantage of using a taxonomy as the basis for a framework, is that it provides consistency in framing objectives for teaching, and criteria and standards for assessment. However, if this type of taxonomy is rigidly adhered to, it can result in a course involving acquiring a lot of knowledge in early years, with students demonstrating increasingly higher order thinking (such as analysing and evaluating) but learning no new knowledge in later years (Engineering Subject Centre, 2000-2007).

Oliver, Dobele, Greber and Roberts (2004) found that when they mapped the levels of outcomes in courses, there was no evidence to support the assumption that students could only achieve the higher levels of Bloom's taxonomy if they achieved the lower levels (such as knowledge) first. Buck and Stucki (2000) found that Bloom's taxonomy did not reflect the processes of software development and needed to be applied in the *reverse*. They also were unable to apply it to higher levels of study – they asked 'the community to help us fill in more details of how we can effectively build from the lower to higher cognitive levels' (p79).

Other authors have based their frameworks on graduate attributes (university-wide graduate outcomes). For example, Whitfield (2003) describes a process at Slippery Rock University in which computer science degree outcomes were placed into four groupings and mapped to graduate attributes. These groupings were: 'problem-solving and decision making', 'critical thinking and analytical reasoning', 'communication and interpersonal skills', 'ethical and professional responsibilities' (p214). The emphasis for teaching and assessing each of these in a subject was determined by a rating of 1-3. The Faculty also formalized and stipulated, instruction strategies and assessment methods to staff that aligned with the groupings. Students at Slippery Rock University were provided with criteria sheets in advance, but these were inconsistently designed — some were checklists to tick, others were descriptions of expectations at four different standards. An evaluation of their framework found there were far too many outcomes to teach and assess, and the 'communication and interpersonal skills' group of outcomes urgently needed revision (Slippery Rock University, 2006).

Matsuura (2006) took a different approach to create a process-product-people framework based on a software development method (requirements analysis, system analysis, system design, implementation, testing and meets user requirements). His framework stipulates products under each part of the software development method. For example, under Implementation (p165): 'lists of source codes, source codes, installation guides, user manual' rather than the qualities of these to be assessed. This framework is not applied across year levels. The criteria that are itemized further for assessment (for example, under Process, p166: 'validity of the selection and construction of tasks for the work plan') appear unrelated to the software development method. Students are given the criteria but these are not accompanied by described standards to show how an overall grade is determined. This indicates implicit standards are used by teachers to judge the quality of student work.

3 Developing the Teaching and Assessment of Software Development framework

We undertook the following steps to develop the framework within a criterion-referenced approach to teaching and assessment.

1. Initially each section of the outlines for the key software development subjects in the IT undergraduate and postgraduate degrees were examined (the rationale, aim, objectives, teaching and learning approaches, assessment and resources).
2. Rather than mapping content, we focussed on common verbs, that is, what students are required to 'do' in these subjects.
3. We recognised that these common verbs happened to map to the recognisable stages of software development. They were reflected in assessment tasks and had the possibility of becoming assessment criteria.

We added 'communication' because we and industry believe that students need to be able to communicate effectively in professional practice. Communication is also deemed important by professional bodies and accreditation boards.

3.1 Defining our use of terms

Regardless of the software development model used, they all have recognisable stages that correspond to the terms analysis, design, implementation, testing. There may be a one-to-one relationship between these stages and those of a particular model, or one stage may correspond to multiple steps in a model. A model may require stage(s) to be carried out iteratively. In this paper, each of the terms (analysis, design, implementation, testing) describes a continuum from simple to advanced in relation to concepts and their integration, and skill development. Depending on the stage being taught, students may be given part or all of some of the other stages. For example, in an assessment task that focuses on design and implementation, students could be given a complete analysis and a partial test plan or suite. The components of the TASD framework are Analysis, Design, Implementation, Testing and Communication. These are used to frame objectives and align them with criteria for assessment tasks. They are also in alignment with the requirements of computing degrees set out in

Computing Curricula 2005 and its companion discipline specific volumes (Cassel et al. 2008; Gorgone et al. 2002; Lunt et al. 2008; Shackelford et al. 2006). Our definitions of the TASD framework components are as follows:

Analysis: this term involves understanding the components of a problem by, for example, defining feasibility and scope, and determining the user requirements

Design: this term includes interpreting and/or producing a solution to a problem—this ranges from high level architectural design to detailed algorithmic design

Implementation: this term includes coding of the solution, coding style and standards, program documentation, efficiency and data structure choices

Testing: this term includes testing the correctness of the design and the functionality and efficiency of the program; done by designing test plans and executing the tests

Communication: this term involves oral and written modes of expression using the language and genres of professional practice, for example, report writing, oral presentation, and use of industry templates for documentation.

3.2 Determining validity and applicability

As an initial validation of the framework, we applied it to a series of subjects across year levels as they were currently taught and assessed. This was convenient because the subjects being taught (by authors Thomas and Corney) spanned four year levels and emphasized different components of the framework. Part of the validation was to examine the developmental progression of the framework components. After this initial validation, we then checked all the other software development subjects taught at QUT. Naturally each subject places different emphasis on different components, determined by the amount of time allocated to the teaching and assessment of the components.

4 Implementing the framework

4.1 Determining the assessment criteria

We implemented the TASD framework by using it to devise criteria sheets for assessment tasks from different subjects. Table 1 lists the aspects of the criteria that we considered important in assessing students' knowledge and skills, and which can be identified in their work.

These aspects are sufficiently fine-grained without overwhelming students in the minutiae and making assessment tedious. This list is therefore not exhaustive. Nevertheless, it serves as a useful starting point when devising tasks, their criteria and associated descriptions of standards. The QUT assessment policy requires that we describe five standards referenced to criteria. For the purposes of this paper, these standards (achievement levels) are represented as A to E, where A is the top grade, D is a passing grade and E is a failing grade. On a criteria sheet these standards are represented by bullet points, termed descriptors. In criterion-referenced assessment, students demonstrate that they meet a described standard in order to be awarded a particular grade. In contrast, norm referencing relies on a preset distribution of the number of students to be awarded each grade, regardless of the standard of their work.

Criterion	Aspects that can be assessed
Analysis	clarity and structure of system overview, quality of use case diagrams and descriptions, meaning and measurability of non-functional requirements, identification of actors, …
Design	use of a design language to describe a design, identification of classes, quality of class descriptions, cohesion, coupling, quality of architecture, internal logic and data flow, correspondence between the design and analysis model, coverage of requirements, evaluation of the design, plausibility of the implementation and testing scheduling, …
Implementation	completeness of task, code quality, commenting, functionality, robustness, quality of the user interface, …
Testing	code coverage, requirements coverage, handling of boundary conditions, extent of condition testing, description of input and output, acceptance testing environment description, repeatability, system testing, unit testing, acceptance testing, …
Communication	*Written*: adherence to English conventions, clarity of writing, level of detail, narrative flow, conformance to industry standard templates,… *Oral*: contributions to team meetings, coherence and clarity of presentation, content and structure, delivery, timing, interaction with audience, …

Table 1: Aspects of the TASD framework criteria

4.2 Levelling expectations

4.2.1 Levelling across year levels

As shown in table 1, within the *implementation* criterion, one of the typical aspects that can be assessed is code quality. The following examples use this aspect to illustrate how levelling can be achieved between a first year and a Masters subject.

In the first year software development subject (Example 1), students are taught the "basics" of code quality, including such things as naming of identifiers and commenting to build good habits so they produce readable code. By Masters level, we expect students to produce readable *and* efficient code. Example 1 above is an excerpt from a criteria sheet for a first year assessment task where the students were required to write a program. This criteria sheet illustrates our expectations for code quality and two of the five achievement levels. In this task, the readability of the code is emphasized by assessing it in detail.

Readability of code is also important for more advanced programmers. However, at a higher level, this code quality aspect should have been already learnt and it is expected that the student will produce readable code. So, now a "higher level" of code quality can be assessed.

Implementation	You have written code that:	
	Standard A	**Standard D**
Code Quality Quality issues in the Style Guide include: layout, indentation, spacing, identifier naming	• conforms 100% to the style guide	• has some inconsistencies and/or problems with two code quality issues
Commenting As per Style Guide, workshops and lectures	• has a comment for every class, method and instance variable where appropriate • has comments which give a good description of function (including parameter and return values for methods) • includes pre and post conditions for all methods • uses XML style commenting	• has a comment for most classes, a few methods and instance variables where appropriate • has comments which give a fair description of function, these may not include parameter and return values for methods • does not include pre and post conditions

Example 1: Extract of criteria sheet, first year Object Oriented Programming

Example 2 is from a programming assessment task for Masters level students. Note that all of the code quality expectations from the first year criteria sheet have been subsumed into one descriptor "is readable" in the Masters' compiler task. The emphasis in the Masters' task is on more advanced code quality aspects, such as efficiency of code, which would not be assessed in first year.

Implementation	You have written code that:	
	Standard A	**Standard D**
Code Quality • Choice and use of code constructs and data structures. • Readability (comments, variable names, layout, etc.)	• uses code and data structures which are efficient with regard to memory use and execution speed • is readable	• uses code and data structures which result in some inefficiencies with regard to memory use and execution speed • is mostly readable

Example 2: Extract of criteria sheet, Masters' subject Compiler Construction

Example 3 (on the right) illustrates how the *analysis* criterion is assessed across year levels. The first subject is the prerequisite for the second one. As in previous examples, levelling is achieved by aspects of the criterion in the second year subject subsuming and extending some of the first year aspects.

4.2.2 Levelling within a year level

Example 4 demonstrates the levelling of expectations within the *design* criterion for the first year second semester subject Modelling Analysis and Design. We

Modelling Analysis and Design (first year)	
Generic task description: • students do a simple analysis of a large problem • derive a three-tiered architecture • design a detailed solution for the middle tier	
Aspects of Analysis	**Standard for an A**
Quality of use case diagrams	• identified the main, and some minor, actors which were all external entities that interact directly with the system. • accurately summarised the main system features and which actors use them.
Quality of use case descriptions	• provided accurate, detailed and concise descriptions of the main system features that captured all the information about using the system from the actors' perspectives.
Software Engineering Studies (second year)	
Generic task description: • students analyse a large problem • design a solution • implement and verify the solution	
Aspects of Analysis	**Standard for an A**
Quality of use case diagrams and descriptions	• a clear and structured overview is provided of system functional and environment requirements. • your use case descriptions unambiguously and concisely capture all important information about using the system, from a user's perspective.
Meaning and measurability of non-functional requirements	• non-functional requirements are meaningful and measurable constraints, as negotiated.

Example 3: Extract of criteria sheets from two different year levels showing highest grade for the analysis criterion

have achieved effective levelling by the judicious use of adverbs and adjectives, combinations of these, and depending on the standard, additional expectations within the descriptors. We have extracted a descriptor for the "Quality of Class Diagrams" aspect and identified how this levelling has been achieved, by italicizing and underlining the differences. Additional expectations are in bold.

Standard A

Structured the design *with packages* of logically related classes and with ***minimal*** **dependencies between packages**

Standard D

Structured the design *with a few packages* of loosely-related classes

Example 4: descriptor comparison for first year Modelling Analysis and Design

4.3 Adapting criteria to suit task design

4.3.1 All relevant criteria do not have to be assessed in each task

The particular criteria that are used in a criteria sheet should suit the task and reflect the objectives of the

subject. This means that what you assess should be what you have taught—a fundamental principle of alignment (Biggs, 2003). It is not necessary to use every criterion in every task as long as the assessment plan has a balance across the criteria. The criteria you select will depend upon the emphases, not only of the subject, but also of the particular task being assessed. For example, a *task* might emphasize *implementation*, while the *subject* emphasizes the three criteria of *implementation*, *design* and *testing*; the assessment tasks for the whole subject would, together, assess all three of these criteria.

4.3.2 Aspects of a criterion may be subsumed

Aspects introduced in one task may be subsumed in subsequent tasks when the students are aware of implied expectations and should have acquired the necessary skills. To illustrate this, we use the subject Object Oriented Programming as an example. Students work on a "large" (for first year) program throughout the semester, from which a number of assessment tasks are submitted. In task 1, 'commenting' is assessed separately from code quality, while in task 2, 'commenting' is deliberately subsumed, as it is part of code quality. This is because task 1 has scaffolded the development of student skills so that in task 2 students should grasp what is expected for 'commenting'.

Task 1	**Task 2**
Involves writing test plans and code. Students are given a design and are required to write the declarations to provide a shell for the program. Then they implement a very small part of the program logic.	Builds on task 1 by requiring students to implement the rest of the program logic to complete the project (after feedback has been provided on students' response to task 1)
Criteria: **Implementation** **Testing**	**Criterion:** **Implementation**
Aspects of the criteria: **Implementation:** • Completeness of the task (quantity and correctness of code- how much has been attempted by the student and how much is correct) • Code Quality (assesses the readability of the code the same as in example 1 (section 4.2), • Commenting (see example 1, section 4.2) assessed separately in this task	*Aspects of the criterion* **Implementation:** • Functionality (quantity and correctness of code that meets the supplied specification) • Code Quality (assesses the readability of the code the same as in example 1 (section 4.2), but subsumes commenting)
Testing: • unit test plans (quality and completeness)	

Example 5: Criteria for related assessment tasks for Object Oriented Programming

4.3.3 The aspects of criteria and descriptors of standards can be reused in different tasks

Reuse of aspects is possible when tasks are similar, regardless of the year level. Examples 1 and 2 show how an aspect of implementation (code quality) can be reused across different subjects. It is also possible to reuse an aspect within the same subject in different assessment tasks. Note that the descriptors of the standards need not be the same. If the tasks are similar and the year level is the same it is also possible to reuse the descriptors. For example an aspect of communication is "adherence to English conventions". The descriptors for this aspect are generic enough to apply to any task requiring communication within a year level. Adherence to English conventions aspect may be reused at other year levels but the descriptors would have to be altered, that is increasing or decreasing expectations of students to reflect the level of difficulty of the subject.

4.3.4 Criteria can be combined into one criterion

The Framework allows you to combine criteria for an assessment task. For example, in Masters' level Compiler Construction, one of the assessment tasks involves the production of a Finite State Automata diagram. This task involves *analysis* and *design*, so the criteria are combined into a single criterion on the criteria sheet (see example 6). We chose not to assess analysis separately from design, as producing an analysis artefact would not be an authentic task in industry practice.

Generic task description: This task requires the student to produce a design artefact in the form of a Finite State Automata diagram.		
Criterion	You have analysed the functionality required to design a FSA that:	
	Standard A	**Standard D**
Analysis and Design	• is deterministic • has all the required states and transitions to recognise all the symbols of the language • has no unnecessary states	• may be deterministic • has the required states and transitions to recognise more than half of the symbols of the language

Example 6: Extract of criteria sheet, Compiler Construction.

4.3.5 Aspects should be selected to suit task goals

The second year subject Software Engineering Studies requires students to work on a large project over a semester (typically 13 weeks). There are several milestones and each of these is an assessment task with its own criteria sheet.

Example 7 shows that all of the criteria are assessed on this project – some aspects of the criteria appear on more than one criteria sheet for different parts of the project (e.g. clarity of structure of system overview, use of a design language to describe a design, conformance to industry standard templates).

Generic task description: This task requires students to work in groups to analyse, design, implement, test and deliver a software product.	
Criterion	**Aspects that are assessed**
analysis	clarity and structure of system overview, quality of use case diagrams and descriptions, meaning and measurability of non-functional requirements
design	use of a design language to describe a design, identification of classes, cohesion, coupling, internal system logic and data flow, level of correspondence between the design and the analysis model, plausibility of the implementation and testing scheduling
implementation	code quality, functionality, quality of the user interface
testing	requirements coverage, description of input and output, acceptance testing environment description, effectiveness of tools usage, repeatability, system testing, unit testing, acceptance testing
communication	*written:* adherence to English conventions, narrative flow, conformance to industry standard templates

Example 7: Aspects of the criteria for Software Engineering Studies large project

5 Comparison of our frameworks to others

The only framework that has some similarities to ours is the product process-people one developed by Matsuura (2006). This is because it is based on a software development process and includes 'communication' (as part of the 'process' and 'people' components). However, its sole purpose is for assessing an individual's problem-based learning when in a group. Students are assessed in terms of what artefacts they have created, not the standard of these against criteria. In comparison with the frameworks referred to earlier in the literature review, our framework has many advantages. It serves the purposes of teaching and assessment and it is based on generic stages of software development. This makes the framework coherent and logical, and sufficiently generic to be used by others. It does not involve assessing a myriad of objectives/outcomes, or use a particular taxonomy of learning objectives—this makes it more manageable and practical. Our framework follows the principles of criterion-referenced assessment by using the components to frame objectives for subjects and align them with criteria for assessment tasks. It also shows how to describe different standards within and across year levels. This means that expectations are clear and explicit so that students have more control over their own learning.

One of the most important advantages of the framework is that teachers no longer have to use an ad hoc approach to the creation of criteria. We have provided a logical starting point with five criteria that can easily be contextualized to suit particular tasks using the aspects we have listed in table 1. This makes the framework adaptable, regardless of what part of software development is being taught and assessed, and it gives teachers control over how they teach and assess. The list of aspects will evolve as teachers explore different types of assessment. The framework, however, is not an automatic generator of assessment tasks or criteria sheets—teachers still have to devise tasks and make decisions on what criteria best suit these. A further advantage is that the framework can be used to improve subject and course/ degree design by ensuring that teaching and assessing are fully aligned.

6 Evaluation

In this section, initial qualitative and quantitative data is presented. To date, TASD has been implemented in nine subjects and been used to redesign the Bachelor of Information Technology degree/course.

6.1 Qualitative data

Figure 1 illustrates the remarkable difference in student satisfaction about the difficulty level of assessment in a subject across four semesters (2006, two in 2007 and one in 2008). The TASD framework was implemented by one of the authors for the middle two semesters, with another teacher delivering it in semester four. A key feature of the framework is the use of criteria sheets, by tutors to grade students' work, and by students to judge the quality of their own work before submitting it (see section 6.1.1 for a sample of comments).

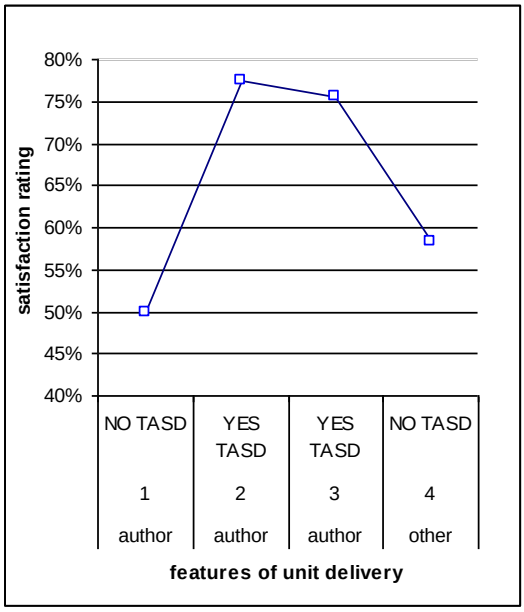

Figure 1: Student satisfaction rating of assessment difficulty for first year Modelling Analysis and Design

6.1.1 Selection of comments about using criteria sheets

Students

The criteria sheets helped me to know what I needed to do to get a good result.

I like getting criteria sheets for assignments so I know what is required, they clear up expectations for the assignment.

Criteria sheets helped me to manage my time.

Tutors

I need the discussion about a criteria sheet to understand how to interpret it and use it for marking.

I like having a standard to use to judge the quality of assignments.

Once I get used to a criteria sheet it speeds up my marking.

A good criteria sheet makes it easier to talk to students about marking and what is required for an assignment.

The TASD framework informed the approach to the redesign of the Bachelor of Information Technology (BIT) degree in 2008. This resulted in cohesion between subjects which were no longer topic-driven but process-driven, forming a better foundation for industry practice than the previous traditional approaches. The type of content remained the same as did academic rigour. 2009 was the first year of this degree. Table 2 shows students' satisfaction ratings of three features of the revised core subjects.

Revised Core Subjects	Workload	Difficulty	Relevance
1	89.90%	93.90%	79.80%
2	61.70%	77.70%	84%
3	84.40%	86.20%	63.30%
4	94.40%	84.90%	97.60%

Table 2: % satisfied students–ratings of first semester core subjects in the revised Bachelor of IT degree

6.2 Quantitative data

In 2009, the attrition rate at the end of first semester was a significant improvement on previous years (table 3). The overall pass and fail rates in first semester subjects also improved in 2009 (table 4). It is not possible to directly compare the pass and fail rates for individual subjects from 2008 to 2009 as the redesign of the BIT has resulted in the first semester core subjects being very different to previous versions. Table 4 does line up the subjects from 2008 with their closest replacement in 2009 but the teaching approaches, content and structure of the subjects in the revised BIT are substantially different.

Degree & Number of Students	Degree Structure	Attrition Data: % of Enrolled Students	
		end semester 1	end week 3 semester 2
2009 (n=208)	revised	9%	9%
2008 (n=179)	original	35%	34%

Table 3: Comparison of attrition data for Bachelor of IT commencing students 2008-2009

2008 Semester 1 Results			2009 Semester 1 Results		
Core Subjects	original (n=179)		Core Subjects	revised (n=208)	
	% pass	% fail		% pass	% fail
1	81%	19%	4	96%	4%
2	95%	5%	3	94%	6%
			1	94%	6%
4	76%	24%			
5	85%	15%	2	85%	15%

Table 4: Comparison of pass and fail rates for the original and revised core units of the Bachelor of IT

One of the first subjects in which TASD was applied was Modelling Analysis and Design. Figure 2 shows that there was not a large difference in student results when comparing the two semesters in which criteria sheets were used, versus the previous and following semesters. The one important difference is that in 2006/2 the cut-off for a grade of A was normalised. Consequently there is a higher percentage of A grades and a lower percentage of B grades than would have been the case if the cut-offs were not altered. If normalisation had not taken place in 2006/2 only 2% of students would have achieved a grade of A and 24% would have achieved a grade of B; resulting in a line very similar to 2007/1 and 2007/2. As there is no significant difference in student results between the semesters that used or did not use criteria sheets it may be assumed that the use of criteria sheets does not impact on student results. Section 6.1 noted that student satisfaction with the difficulty of the assessment was markedly increased in the semesters in which criteria sheets were used. Students, and tutors, commented positively about criteria sheets in their feedback on the subject. Consequently we would recommend the use of criteria sheets, and TASD to support their development, as they seem to lead to increased student (and tutor) satisfaction without any apparent negative affects on student results.

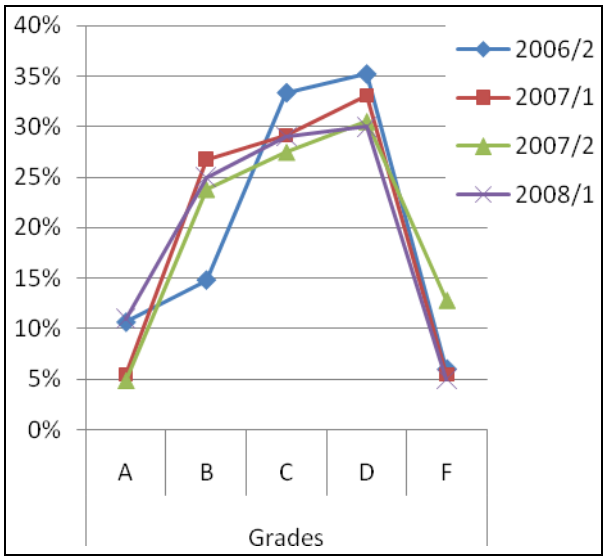

Figure 2: Comparison of results for Modelling Analysis and Design (2006/2 – 2008/1)

7 Conclusion

We devised the TASD framework by taking a holistic approach in which we considered the whole picture of curriculum and assessment across year levels. This meant determining the content and assessment similarities of *all* software development subjects, not just isolated ones. The biggest challenges were to ensure that the framework was practical and adaptable, true to the discipline and aligned with international computing science curricula. Initially, we implemented it in four software development subjects in 2006-7. Resources developed during that time included revised subject outlines, a table of aspects of criteria suitable for assessment at different year levels and a suite of criteria sheets (extracts of which are

presented in this paper). The framework has since been presented to faculty and used to revise the structure of a degree (2009 was its first year).

In section 6 we stated that the introduction of criteria sheets improved student satisfaction without markedly affecting student results. This was an expected outcome of the project as we were trying to explicitly show students the alignment of assessment with learning objectives and what standards they had to achieve to be awarded certain grades. (An untested side effect is the hope that this will help students become more self evaluative.) In the experience of the one author who has continued in a teaching position at QUT, I have found that using the TASD framework has helped with the design and setting of expectations for assessment tasks. Linking criteria to learning objectives helps focus on what activities are required for an assessment task so that students achieve the subject's learning objectives.

When redesigning the Bachelor of IT degree at QUT the design team purposely chose not to follow common implementations of the ACM/IEEE model curricula with a standard CS1, CS2, … sequence of subjects (Cassel et al. 2008; Gorgone et al. 2002; Lunt et al. 2008; Shackelford et al. 2006). The TASD framework provided a mechanism to ensure that our sequence of subjects achieved our goals and covered the important knowledge areas from the model curricula without being constrained by standard content delivery approaches.

We believe that our framework has advantages over existing frameworks. It reflects the processes of software development, has a logical and consistent whole of degree approach, is adaptable between subjects and across year levels, and makes assessment expectations explicit through the use of criteria sheets. Initial qualitative and quantitative data indicates that implementing this framework improves student satisfaction about assessment difficulty, workload and relevance of subjects, as well as reducing attrition rates.

8 References

Biggs, J. B. (2003) *Teaching for quality learning at university*, Buckingham, UK: SRHE and Open University Press.

Bloom, B. & Krathwohl, D. 1956. *Taxonomy of Educational Objectives.* New York: McKay & Co.

Box, I. 2004. Object-oriented Analysis, Criterion Referencing, and Bloom. *Proceedings of the 6th Australasian Computing Education Conference (ACE2004)*, Dunedin, NZ. Volume 30. pp. 1-8.

Buck, D. & Stucki, D. 2000. Design Early Considered Harmful: Graduated Exposure to Complexity and Structure Based on Levels of Cognitive Development. *Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education, 2000*, Austin, Texas. pp. 75-79.

Burgess, G. 2005. Introduction to Programming: Blooming in America. *Journal of the Consortium for Computing Sciences in Colleges*. Volume 21, Number 1 (Oct.). pp. 19-28.

Cassel, L., Clements, A., Davies, G., Guzdial, M., McCauley, R., McGettrick, A., Sloan, R., Snyder, L, Tymann, P. and Weide, B. 2008. *Computer Science Curriculum 2008: An Interim Revision of CS 2001*. ACM and IEEE.

Daniels, M., Berglund, A., Pears, A. & Fincher, S. 2004. Five Myths of Assessment. *Proceedings of the 6th Australasian Computing Education Conference (ACE2004)*, Dunedin, New Zealand. Volume 30. pp. 57-61.

Engineering Subject Centre. 2000-2007. Levels in Module Descriptions. London: *The Higher Education Academy*. http://www.engsc.ac.uk/er/theory/levels.asp (accessed 6 June 2007)

Gorgone, J., Davis, G., Valacich, J., Topi, H., Feinstein, D. and Longenecker, H. Jr. 2002. *Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems*. Association for Information Systems.

Lister, R. and Leaney, J. 2003. Introductory Programming, Criterion-Referencing, and Bloom. *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education, 2003*, February 19-23, Reno Nevada USA, p. 203.

Lunt, B., Ekstrom, J., Gorka, S., Hislop, G., Kamali R., Lawson, E., LeBlanc, R., Miller, J. and Reichgelt, H. 2008. *Information Technology 2008: Curriculum Guidelines for Undergraduate Degree Programs in Information Technology*. ACM and IEEE.

Matsuura, S. 2006. An Evaluation Method of Project Based Learning on Software Development Experiment. *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education, 2006*, Houston, Texas. pp. 163-167

Michael, M. 2000. Fostering and Assessing Communication Skills in the Computer Science Context. *SIGCSE Bulletin*. pp.119-123

Oliver, D., Dobele, T., Greber, M., & Roberts, T. 2004. This Course has a Bloom Rating of 3.9. *Proceedings of the 6th Australasian Computing Education Conference, (ACE2004)*, Dunedin, NZ. Volume 30. pp. 227-231.

Shackelford, R., Cross II, J., Davies, G., Impagliazzo, J., Kamali, R., LeBlanc, R., Lunt, B., McGettrick, A., Sloan, R. and Topi, H. 2006. *Computing Curricula 2005: The Overview Report*. ACM and IEEE.

Slippery Rock University. 2006. Computer Science Department Assessment Report. http://cs.sru.edu/~whit/assessment/Report06.ppt (accessed 10 October 2007)

Watson, G., Morgan, C., McKenzie, T., Roberts, D., & Cochrane, K. 2002. Meeting the Challenge of Positioning Undergraduate Units of Study at the Appropriate Levels. *Higher Education Research and Development*. pp. 704-712.

Whitfield, D. 2003. From University Wide Outcomes to Course Embedded Assessment of CS1. *Journal of the Consortium for Computing Sciences in Colleges*. Volume 18, Number 5, pp. 210-220. Sample rubrics: http://cs.sru.edu/~whit/assessment/ (accessed 10 October 2007)